红黑树(Red-black Tree) & 伸展树(Splay Tree)

伸展树

是一种二叉排序树，它能在O(logn)内完成插入、查找和删除操作
总是将刚刚访问过的节点通过左旋/右旋操作，在保持二叉排序树的特性下，移动至树根位置
应用举例:
删除一颗二叉排序树中所有大于u小于v的节点(将u调至根节点，v调至u的右节点)
注意点:
为了达到最高效率，伸展树需要两层两层伸展(最后一次伸展可以按一层来伸展)
对于LR和RL采用自下而上旋转
对于LL和RR采用自上而下旋转

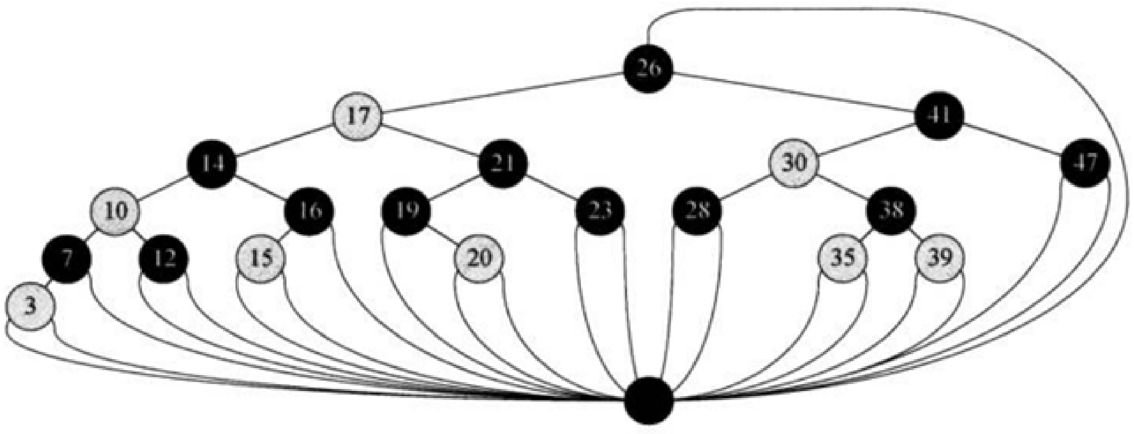
红黑树

每个结点要么是红色，要么是黑色
根结点是黑色
每个叶结点，即空结点(NIL)是黑色
如果一个结点是红色，那么它的俩个儿子都是黑色
对每个结点，从该结点到其子孙结点的所有路径上包含相同数目的黑结点

推论:
- 高度h<=2*log(n+1)
- 红黑树是一颗近似平衡的二叉搜索树

把黑父节点左右两边的红子节点合并成一个超级节点，就成了一棵4阶B树

带哨兵的红黑树



旋转

在 LEFT-ROTATE 的伪代码中，假设 $x.right \neq T.nil$ 且根结点的父结点为 $T.nil$。

```
LEFT-ROTATE(T, x)
y = x.right                 // set y
x.right = y.left            // turn y's left subtree into x's right subtree
if y.left ≠ T.nil
    y.left.p = x
y.p = x.p                   // link x's parent to y
if x.p == T.nil
    T.root = y
elseif x == x.p.left
    x.p.left = y
else x.p.right = y
y.left = x                  // put x on y's left
x.p = y
```

图 13-3 给出了一个 LEFT-ROTATE 操作修改二叉搜索树的例子。RIGHT-ROTATE 操作的代码是对称的。LEFT-ROTATE 和 RIGHT-ROTATE 都在 $O(1)$时间内运行完成。在旋转操作中只有指针改变，其他所有属性都保持不变。

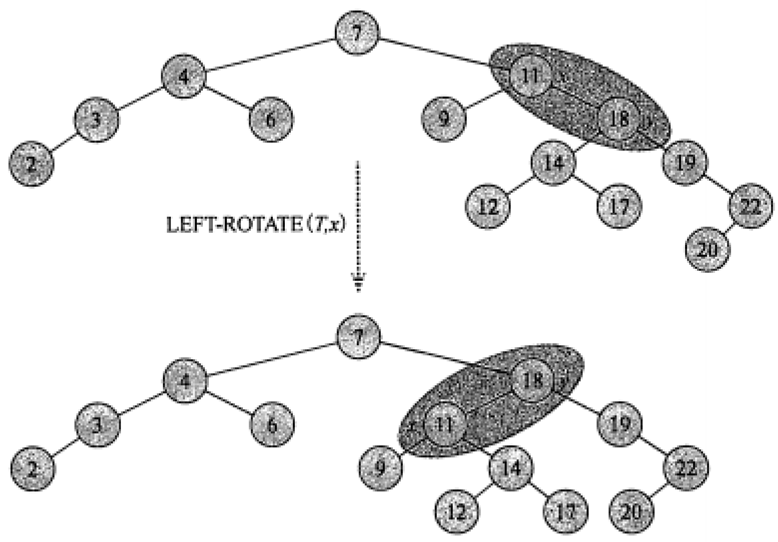


图 13-3 过程 LEFT-ROTATE(T, x)修改二叉搜索树的例子。输入的树和修改过的树进行中序遍历，产生相同的关键字值列表

插入

我们可以在 $O(\lg n)$时间内完成向一棵含 n 个结点的红黑树中插入一个新结点。为了做到这一点，利用 TREE-INSERT 过程(参见 12.3 节)的一个略作修改的版本来将结点 z 插入树 T 内，就好像 T 是一棵普通的二叉搜索树一样，然后将 z 着为红色。(练习 13.3-1 要求解释为什么选择将结点 z 着为红色，而不是黑色。)为保证红黑性质能继续保持，我们调用一个辅助程序 RB-INSERT-FIXUP 来对结点重新着色并旋转。调用 RB-INSERT(T, z)在红黑树 T 内插入结点 z，假设 z 的 key 属性已被事先赋值。

```
RB-INSERT(T, z)
y = T.nil
x = T.root
```

```
3   while x ≠ T.nil
4       y = x
5       if z.key < x.key
6       x = x.left
7       else x = x.right
8   z.p = y
9   if y == T.nil
10      T.root = z
11  elseif z.key < y.key
12      y.left = z
13  else y.right = z
14  z.left = T.nil
15  z.right = T.nil
16  z.color = RED
17  RB-INSERT-FIXUP(T, z)
```

过程 TREE-INSERT 和 RB-INSERT 之间有 4 处不同。第一，TREE-INSERT 内的所有 NIL 都被 $T.nil$ 代替。第二，RB-INSERT 的第 14～15 行置 $z.left$ 和 $z.right$ 为 $T.nil$，以保持合理的树结构。第三，在第 16 行将 z 着为红色。第四，因为将 z 着为红色可能违反其中的一条红黑性质，所以在 RB-INSERT 的第 17 行中调用 RB-INSERT-FIXUP(T, z)来保持红黑性质。

RB-INSERT-FIXUP

```
RB-INSERT-FIXUP(T, z)
while z.p.color == RED
    if z.p == z.p.p.left
        y = z.p.p.right
        if y.color == RED
            z.p.color = BLACK                 // case 1
            y.color = BLACK                   // case 1
            z.p.p.color = RED                 // case 1
            z = z.p.p                         // case 1
        else if z == z.p.right
                z = z.p                       // case 2
                LEFT-ROTATE(T, z)             // case 2
            z.p.color = BLACK                 // case 3
            z.p.p.color = RED                 // case 3
            RIGHT-ROTATE(T, z.p.p)            // case 3
    else(same as then clause
            with "right" and "left" exchanged)
T.root.color = BLACK
```

图 13-4 给出一个范例，显示在一棵红黑树上 RB-INSERT-FIXUP 如何操作。

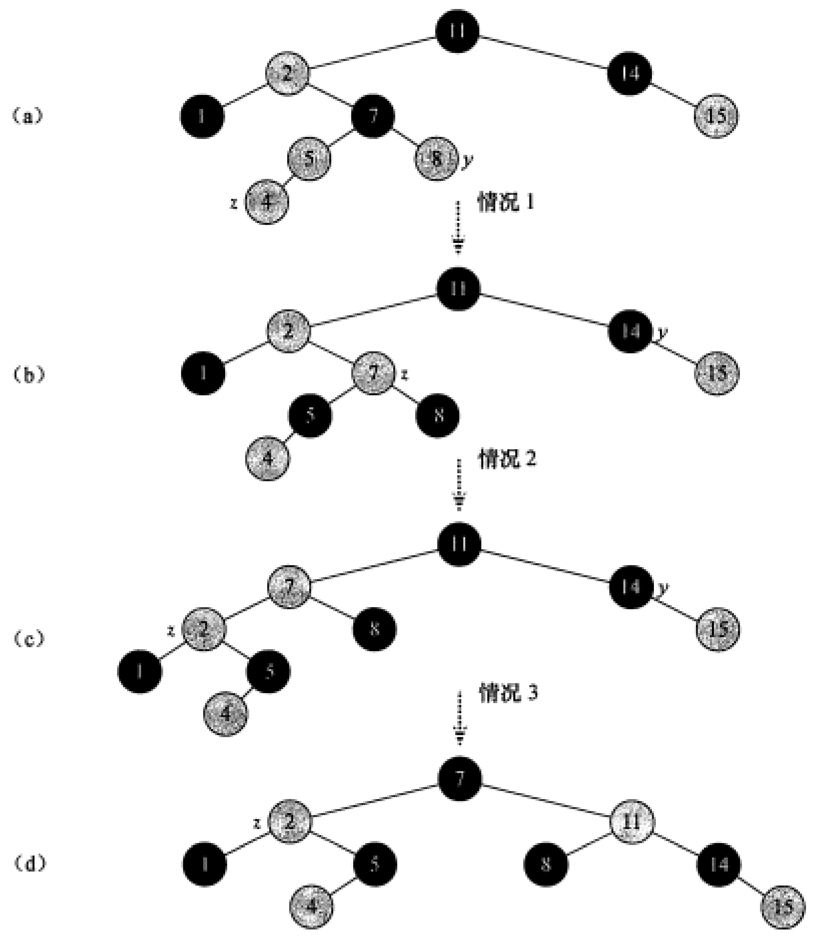


图 13-4 RB-INSERT-FIXUP 操作。(a)插入后的结点 z。由于 z 和它的父结点 $z.p$ 都是红色的，所以违反了性质 4。由于 z 的叔结点 y 是红色的，可以应用程序中的情况 1。结点被重新着色，并且指针 z 沿树上升，所得的树如(b)所示。再一次 z 及其父结点又都为红色，但 z 的叔结点 y 是黑色的。因为 z 是 $z.p$ 的右孩子，可以应用情况 2。在执行 1 次左旋之后，所得结果树见(c)。现在，z 是其父结点的左孩子，可以应用情况 3。重新着色并执行一次右旋后得(d)中的树，它是一棵合法的红黑树